A primer on binary-arithmetic rounding

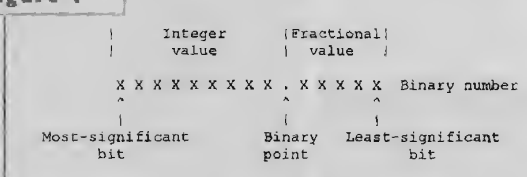
Tom Balph, Motorola SPS, Tempe, AZ

As digital communications and data compression/decompression proliferate, signal-processing functions grow in importance. Whether you're dealing with hard-wired logic or programmable engines, an understanding of binary-arithmetic rounding is important in getting correct and consistent results. Before we discuss rounding, consider a binary number (**Figure 1**).

At first glance, rounding seems a simple matter. However, several variations on rounding exist. Depending on the application, you may use one of the following techniques:

- Truncation (round to minus infinity)—This form of rounding ignores any information in the fractional value to the right of the binary point. You discard these bits, leaving the integer value to the left of the binary point unaffected. Truncation is also called round to minus infinity because it has the effect of rounding to the more negative number. Truncation is in wide use because it is simple to implement: Just ignore the unused bits.
- Round to plus infinity—This variation is essentially the inverse of truncation. If the fractional value to the right of the binary point is not exactly zero, then you round up (make more positive) the integer value. The implementation is more complex than truncation, because you must test all the fractional bits for the existence of a one and then increment the integer value if you find a one.
- Round to zero—Round to zero applies to 2's complement numbers. (For the case of positive numbers only, round to zero reduces to truncation.) For negative numbers, this rounding technique depends on the fractional value—the existence of any nonzero LSBs causes a round up to a less negative number. The positive-number case is simply truncation. The implementation must consider both the fractional-number value and the sign bit. You increment the integer value only when the sign bit equals one (negative number) and the fractional value is not zero.
- Up-magnitude (round to infinity)—Up-magnitude is the inverse of round to zero and applies to 2's complement numbers. If the number is positive, round up for any fractional number not equal to zero. If the number is negative, round down (truncate) for any fractional number not equal to zero. This algorithm is perhaps most useful for maintaining the largest possible magnitude for digital-to-analog conversion. The technique finds use in recent standards, such as ISO/IEC 11172-3 MPEG audio. Implementation again considers the sign and fractional values: Increment the integer only when the sign bit equals zero (positive number) and the fractional value is not zero.
- Simple round (2's complement round)—Simple round applies to both magnitude-only and 2's-complement numbers. You round up the integer value for all fractional values greater than or equal to half the full-scale value of the fractional number. Half the full-scale value is a one and all zeros at the right of the binary point. For fractional values lower than half full-scale, the integer number remains unchanged (truncation). Implementation is relatively simple in that you can add a one to the number at the bit position directly to the right of the binary point. This action increments the integer for any fractional value equal to or greater than half the fractional full-scale value.
- Convergent round—Convergent round is similar to simple round. The difference has to do with the half-full-scale value of the fractional number. A fractional number greater than half full-scale always causes rounding up of the integer number, a fractional number less than half full-scale causes the integer to remain unchanged, and a fractional number equal to half full-scale causes the integer to round up to the nearest even value. Convergent round is most useful for iterative processes in which cumulative addition causes errors to occur more readily. Implementation requires testing the fractional value, as well as the LSB of the integer number. The integer number increments if the fractional value is greater than half full-scale or if the fractional value equals half full-scale and the integer LSB is one (producing an odd number).

Figure 1



A binary number can consist of any number of bits with the binary point at any bit position. All bits to the left of the binary point are the integer, or most-significant, bits of the number. All bits to the right of the point are the fractional, or least-significant, bits of the number. For 2's complement numbers, the most-significant bit has a negative binary weight and is the sign bit. If no sign bit exists, the number is magnitude only and unsigned.

Listing 1 illustrates the rounding methods using Verilog HDL. A 12-bit number, "x," with the binary point located to the left of Bit 3 serves as the input (yielding an 8-bit integer and a 4-bit fraction). Each of the rounded results are 8-bit integer numbers. Part A of **Listing 1** is the module listing, which defines and exercises the rounding outputs and displays the results. Part B gives the displayed simulation results, which you can use to observe the rounding differences. Be aware that, although this HDL routine is fully synthesizable, the resulting logic may not deliver the best performance or be the minimum configuration. You can download **Listing 1** from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the Software Center to down-

load the file from DI-SIG, #2285.

When you implement rounding, performance can suffer if an additional add occurs, because of the rounding algorithm. At times, however, the logic producing the original number can hide the additional add. As an example, if you use simple round (2's complement round) with a multiplier to round the results, a constant one can appear in the partial-product array (at the proper location), and summing the one along with all the partial products produces no loss in performance. Here, the increment of the integer product is buried in the multiplier-adder array. (DI #2285).

To Vote For This Design, Circle No. 389

LISTING 1—ROUNDING EXAMPLES USING VERILOG HDL

```
A) Module listing

//=============================================================
//  File : round.v

module test;

reg [11:0] x;

// Simple truncation

  wire [7:0] trunc   = x[11:4];

// Round-to-plus-infinity
// Increment the integer when the fraction is not equal to zero.

  wire [7:0] plus_inf = (x[3:0] == 4'h0)? x[11:4] :  x[11:4] + 8'h01;

// Round-to-zero
// Increament the integer when a negative number and the fraction is not equal to zero.

  wire [7:0] zero    = (x[11] & (x[3:0] != 4'h0))? x[11:4] + 8'h01 : x[11:4];

// Up-magnitude round
// Increament the integer when a positive number and the fraction is not equal to zero.

  wire [7:0] up_mag  = (~x[11] & (x[3:0] != 4'h0))? x[11:4] + 8'h01 : x[11:4];

// 2's Complement round
// Add fractional number 4'h8 to original number before using integer

  wire [11:0] temp    = x[11:0] + 12'h008;
  wire [7:0] twos_comp = temp[11:4];

// Convergent round
// Increment the integer when; a) fraction is greater than 4'h8,
// or b) fraction = 4'h8 & integer is odd (lsb = 1).

  reg [7:0] converg;

  always @( x )

    if (x[3:0] > 4'h8)              converg = x[11:4] + 8'h01;
    else if ((x[3:0] == 4'h8) & (x[4])) converg = x[11:4] + 8'h01;
    else                            converg = x[11:4];

// display input and outputs

initial
  begin

  x = 12'h000;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,convorg);

  x = 12'h001;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h800;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,convorg);

  x = 12'h801;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h011;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h811;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h008;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,convorg);

  x = 12'h018;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h808;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,convorg);

  x = 12'h818;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h019;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h819;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h028;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h828;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h029;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  x = 12'h829;
  #10; $display("x = %h trunc = %h plus_inf = %h zero = %h up_mag = %h twos_comp = %h
converg = %h",
              x,trunc,plus_inf,zero,up_mag,twos_comp,converg);

  $finish;

endmodule // test module

//=============================================================

B) Simulation results:


Compiling source file "round.v"
Highest level modules:
test

x = 000 trunc = 00 plus_inf = 00 zero = 00 up_mag = 00 twos_comp = 00 converg = 00
x = 001 trunc = 00 plus_inf = 01 zero = 00 up_mag = 01 twos_comp = 00 converg = 00
x = 800 trunc = 80 plus_inf = 80 zero = 80 up_mag = 80 twos_comp = 80 converg = 80
x = 801 trunc = 80 plus_inf = 81 zero = 81 up_mag = 80 twos_comp = 80 converg = 80
x = 011 trunc = 01 plus_inf = 02 zero = 01 up_mag = 02 twos_comp = 01 converg = 01
x = 811 trunc = 81 plus_inf = 82 zero = 82 up_mag = 81 twos_comp = 81 converg = 81
x = 008 trunc = 00 plus_inf = 01 zero = 00 up_mag = 01 twos_comp = 01 converg = 00
x = 018 trunc = 01 plus_inf = 02 zero = 01 up_mag = 02 twos_comp = 02 converg = 02
x = 808 trunc = 80 plus_inf = 81 zero = 81 up_mag = 80 twos_comp = 81 converg = 80
x = 818 trunc = 81 plus_inf = 82 zero = 82 up_mag = 81 twos_comp = 82 converg = 82
x = 019 trunc = 01 plus_inf = 02 zero = 01 up_mag = 02 twos_comp = 02 converg = 02
x = 819 trunc = 81 plus_inf = 82 zero = 82 up_mag = 81 twos_comp = 82 converg = 82
x = 028 trunc = 02 plus_inf = 03 zero = 02 up_mag = 03 twos_comp = 03 converg = 02
x = 828 trunc = 82 plus_inf = 83 zero = 83 up_mag = 82 twos_comp = 83 converg = 82
x = 029 trunc = 02 plus_inf = 03 zero = 02 up_mag = 03 twos_comp = 03 converg = 03
x = 829 trunc = 82 plus_inf = 83 zero = 83 up_mag = 82 twos_comp = 83 converg = 83
L114 "round.v": $finish at simulation time 160
161 simulation events
```

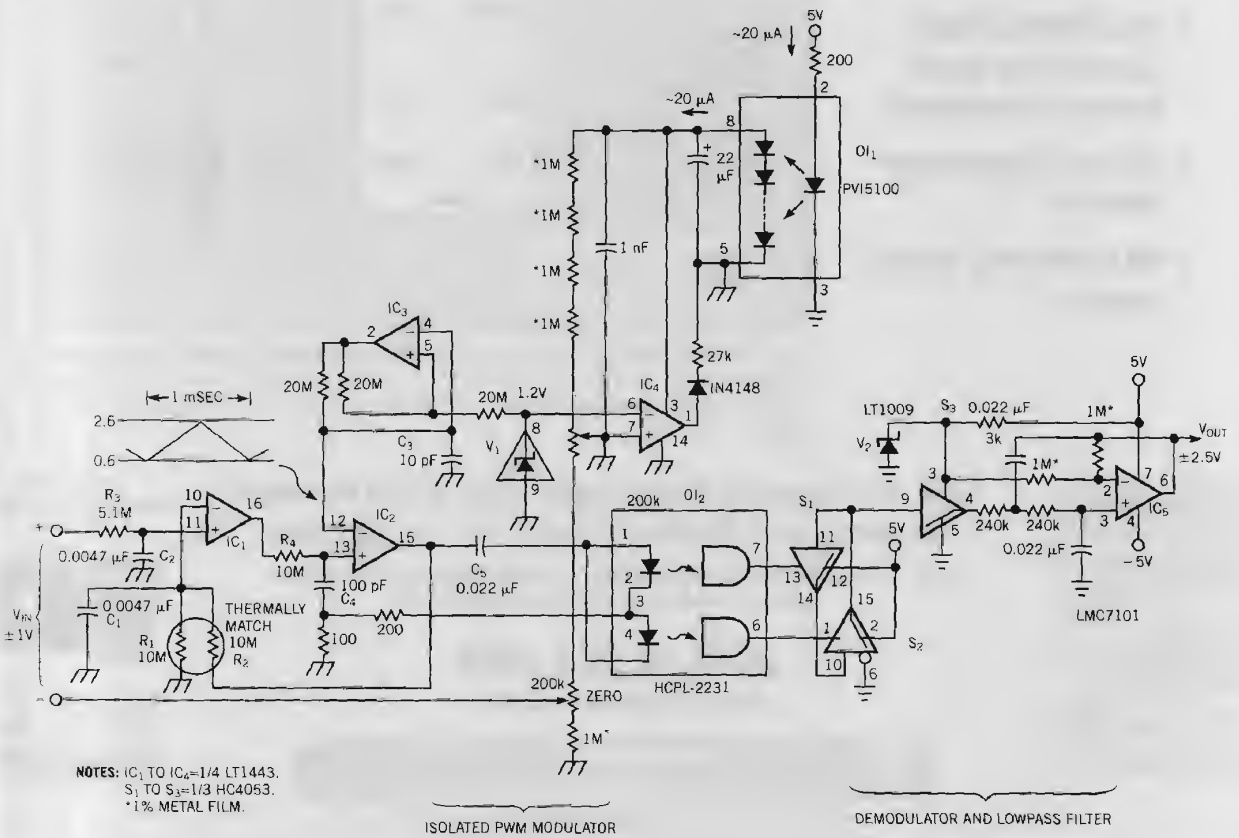
Light powers isolation amplifier

Stephen Woodward, University of North Carolina, Chapel Hill, NC

Self-powered isolation amplifiers, which need no external isolated power supply, provide versatile and convenient interfaces in many applications that require galvanic isolation of the signal source. Examples of such applications include circuits that serve in industrial or medical environments, in which isolation is necessary for noise reduction or safety. You can use a variety of isolated signal-coupling techniques for the signal paths of these amplifiers. Transformers, differential-capacitor, and optoisolator schemes are all popular choices. For the internal isolated power supply, transformer coupling is virtually universal, despite the problems inherent in inductively coupled circuits. These problems include relatively high interwinding stray capacitance and a tendency to couple switching noise into the signal. In contrast, the self-powered amplifier in **Figure 1** is different in that it incorporates optoisolators to effect communication of both signal and power around the isolation barrier.

As in many isolation-amplifier designs, the signal processing in **Figure 1**'s circuit uses PWM. The isolated-modulator front-end circuitry derives from an earlier ADC design and works as follows. IC_1 compares the ±1V filtered input signal with the voltage on C_1. The R_4C_4 time constant smoothes IC_1's output, and IC_2 compares the output with IC_3's approximately 1-kHz triangle waveform. R_1, R_2, and C_1 scale and average the resulting variable-duty-factor square wave and feed the signal back to IC_1. This feedback loop continuously adjusts IC_2's duty factor to maintain equal voltages on C_1 and C_2. In doing so, the feedback forces IC_2's output square wave to track the unique $T_+/(T_++T_-)$ duty factor that maintains balance at IC_1's inputs.

Figure 1



A virtual perpetual-motion machine (when there's light), this self-powered amplifier provides complete galvanic isolation for both power and signal.

C_3 differentiates the IC_2 square wave to provide bipolar drive pulses to the antiparallel LEDs in the high-speed, low-current optoisolator OI_2. In turn, OI_2 produces ground-referred pulses. The rather unusual RS flip-flop formed by cross-connected switches S_1 and S_2 converts these pulses back to a logic-level square wave having the same duty factor as IC_2's output. Demodulation and filtering of the square wave to accurately reproduce the original analog signal occurs through the action of the single-pole, double-throw switch, S_3, which chops the 2.500V V_2 reference voltage according to the $T_+/(T_+ + T_-)$ square-wave duty factor. The lowpass, gain-of-two filter, IC_5, then extracts the dc component of S_3's 0 to 2.5V waveform and scales and offsets it to produce a low-ripple, ±2.5V signal, according to the formula

$$V_O = 2.5 \times T_+ / (T_+ + T_-) = 2.5 \times V_{IN}.$$

Power for the isolated-modulator side of the amplifier comes from OI_1, an International Rectifier (El Segundo, CA) PVI5100 photovoltaic opto IC. Marketed as an isolated MOSFET-gate driver, the PVI5100 can source approximately 20 μA of current at 4V (80 μW), just enough to keep the anorexic LT1443 alive and functional. IC_4 shunt-regulates OI_1's output to provide a stable 4V ratioed against the MAX924's 1.2V±1% internal reference. Overall frequency response is dc to 10 kHz; input impedance is approximately 1 TΩ with less than 1-pA bias. The circuit can thus provide good overall accuracy with high-impedance input sources. You can trim gain and offset errors to zero; the excellent drift specs of the LT1443 maintain the trim over temperature. (DI #2304).

To Vote For This Design, Circle No. 390

Low-cost feedback circuit boosts efficiency

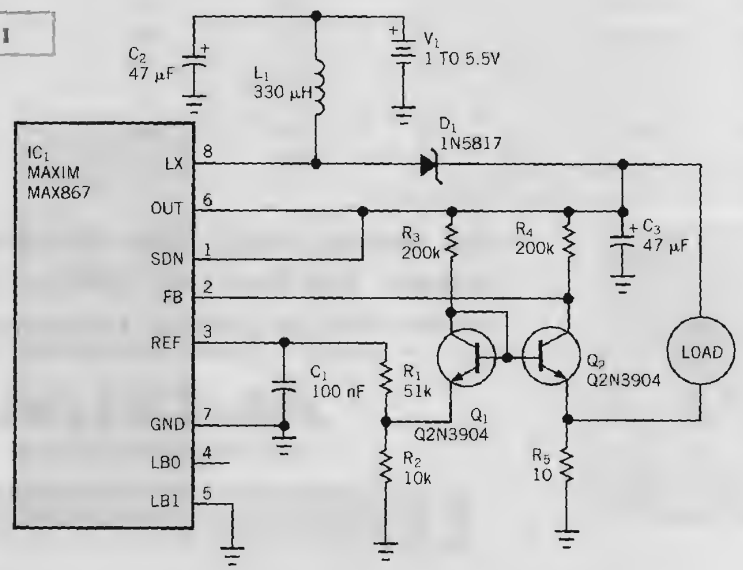
John Guy, Maxim Integrated Products, Sunnyvale, CA

To implement a step-up converter with a current output, designers often simply connect the load in place of the top resistor in a resistive-divider feedback network. The bottom resistor then serves as a current-sense resistor. Though simple, this approach is inefficient. Low efficiency results from the relatively high sense voltages—usually, 1.25V but as high as 2.5V for some ICs. A switch-mode dc/dc converter configured as a 20-mA current source minimizes the efficiency loss by lowering the sense voltage to 200 mV (**Figure 1**). Advantages of this circuit include the factor-of-six gain in efficiency; minimal board area; and readily available, low-cost components. Applications include battery charging, LED drive, and general-purpose current sources.

Resistors R_1 and R_2 form a voltage divider that derives 200 mV from the IC's reference output. This sense voltage connects to one emitter of the current mirror comprising Q_1 and Q_2. Both collectors connect to the output voltage via 200-kΩ resistors. The collector of Q_2 also connects to the IC's feedback pin, and Q_2's emitter connects to the low-side current-sense resistor, R_5. The feedback network appears to the IC's control loop as a common-base amplifier. Selecting a 2N3904 for Q_2 yields sufficient emitter-to-collector gain for the purpose: approximately 80V/V. Moreover, the network's large bandwidth (characteristic of common-base configurations) prevents instability in the IC's control loop. (DI #2307).

To Vote For This Design, Circle No. 391

Figure 1



The use of a current mirror in the feedback loop boosts efficiency and stability in this current-output step-up converter.